AD-A051 448 STANFORD UNIV CALIF DIGITAL SYSTEMS LAB F/G 9/2
DELTRAN PRINCIPLES OF OPERATION: A DIRECTLY EXECUTED LANGUAGE F--ETC(U)
MAR 77 L W HOEVEL DAAG29-76-G-0001
UNCLASSIFIED DSL-TN-108 ARO-12958.3-M NL

1 OF 1
ADA
051448

END
DATE
FILMED
4 -78
DDC

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ARO 12958.3-M

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: ~~Technical Note # 108~~ 12958.3-M
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): DELTRAN PRINCIPLES OF OPERATION: A DIRECTLY EXECUTED LANGUAGE FOR FORTRAN-II
5. TYPE OF REPORT & PERIOD COVERED: Technical Note
6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): Lee W. Hoevel
8. CONTRACT OR GRANT NUMBER(s): DAAG-29-76-G-0011

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Digital Systems Laboratory
Stanford Electronics Laboratories
Stanford University, Stanford, CA 94305

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS:
U.S. Army Research Office-Durham

12. REPORT DATE: March 1977
13. NUMBER OF PAGES: ~~25~~ 29 p.

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): DSL-TN-108

15. SECURITY CLASS. (of this report): unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper describes a novel directly executed language (DELtran) tailored specifically to the FORTRAN source language, EMMY host, and scientific programming. DELtran is "transformationally complete" in that;

(1) Code generation is linear with respect to the number of operators in a FORTRAN program,
(2) Only k DELtran instruction units are needed to represent a FORTRAN statement containing k functional operators,
(3) The space needed to represent a FORTRAN statement approaches N*v+F*k -- where

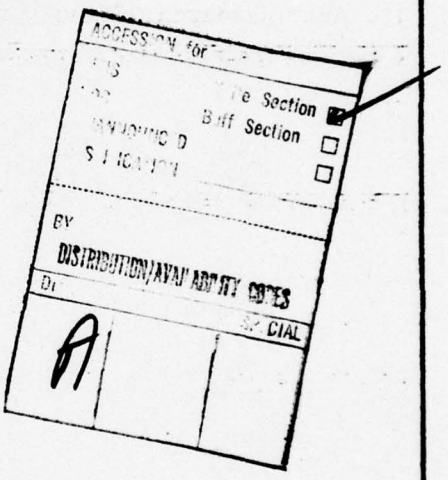
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

408 071

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

v is the number of distinct variables in the statement, and N and F are the least integers such that there are less than 2**N distinct variables and 2**F distinct operators in the relevant scope of definition.

In addition, DELtran is "transparent" in that there is a 1-1 correspondence between DELtran operators and control constructs and FORTRAN operators and control constructs, and "invertible" in that all sensible sequences of DELtran instruction units have a direct FORTRAN analogue.



DELTRAN PRINCIPLES OF OPERATION:
A Directly Executed Language for FORTRAN-II

by

Lee W. Hoevel

March 1977

Technical Note No. 108

DIGITAL SYSTEMS LABORATORY
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305

The work described herein was supported in part by the Army Research Office-Durham under Grant # DAAG-29-76-G-0001.

Digital Systems Laboratory

Departments of Electrical Engineering and Computer Science

Technical Note No. 108

March 1977

DELTRAN PRINCIPLES OF OPERATION:

A Directly Executed Language for FORTRAN-II

by

Lee W. Hoevel

ABSTRACT:

This paper describes a novel directly executed language (DELtran) tailored specifically to the FORTRAN source language, EMMY host, and scientific programming. DELtran is "transformationally complete" in that:

1) Code generation is linear with respect to the number of operators in a FORTRAN program.

2) Only k DELtran instruction units are needed to represent a FORTRAN statement containing k functional operators.

3) The space needed to represent a FORTRAN statement approaches N*v+F*k -- where v is the number of distinct variables in the statement, and N and F are the least integers such that there are less than 2**N distinct variables and 2**F distinct operators in the relevant scope of definition.

In addition, DELtran is "transparent" in that there is a 1-1 correspondence between DELtran operators and control constructs and FORTRAN operators and control constructs, and "invertible" in that all sensible sequences of DELtran instruction units have a direct FORTRAN analogue.

The work described herein was supported in part by the Army Research Office-Durham under Grant # DAAG-29-76-G-0001.

1. Introduction:

DELtran is an intermediate language tailored to a FORTRAN source language, EMMY host machine, and typical community of scientific programmers. Its design is intended to minimize execution phase time and space, subject to the limitations imposed by a one pass compilation that performs only single statement optimization. Our primary objective in synthesizing this language is to demonstrate the practicality of the design principles discussed in Hoevel and Flynn [1], rather than to advance the state of the art in FORTRAN execution, however.

With this in mind, we limited the magnitude of our task by addressing only a subset of the full FORTRAN language (Basic FORTRAN), and ignoring a number of questions relating to a production environment such as higher level data, task, and job management. The resulting design does not preclude extension to features like named COMMON, additional data types and structures, or random access external files. Multiple named COMMON blocks, complex variables, logical variables, relational operators, and simple syntax enhancements like an IF...THEN construct could be implemented merely by changing the compiler or adding a preprocessor. Inclusion of character string data types and dynamic storage allocation features would require altering the executor, which should not be too difficult since it is a table driven interpreter. The instruction unit structure and operand referencing mechanism described below should be compatible with the modifications needed to capture the full FORTRAN language.

General attributes of user communities, high level source languages, and microprogrammable host machines relating to the DEL synthesis problem are discussed elsewhere (Hoevel [2], Flynn [3], Iliffe [4], and Welin [5]), and will not be repeated here. It is instructive, however, to consider those particular features of our experimental system that have had the greatest impact on the design of DELtran.

Source Influence:

The FORTRAN subset of interest here is usually referred to as Basic FORTRAN (Heising [6], McClure [7]). The adjective "basic" is not applied lightly; it is indeed a rudimentary programming language. This turns to our advantage, however, by holding the size of the design problem within reason. Some assumed source language features and restrictions affecting the design of DELtran are:

1) Its name space is entirely static, except for the binding of actual arguments to formal parameters.

2) The natural range for a scope of definition is a procedure specification (i.e., SUBROUTINE or FUNCTION block).

3) Few primitive data types are needed (e.g., only single and double precision forms of fixed and floating point numbers).

4) Unstructured program control is permitted (i.e., DO loops need not be one-in one-out control structures).

5) Parameters are uniformly passed "by reference", although this is equivalent to "by copy value" when expressions are used as actual arguments (this is not required by the standard, but follows the long established IBM tradition).

These observations are extracted from the preliminary ANS specifications for FORTRAN vs. Basic FORTRAN [8]. Immediate implications are: recursive procedure invocation need not be supported; both global and local storage can be statically allocated during compilation; all type checking can be performed during compilation (ignoring parameters to procedures, as is conventional); and program flow analysis can involve arbitrarily complex constructs.

Host Influence:

The basic architecture of the EMMY host and its surrounding laboratory environment are described in Neuhauser [9] and [10]. In general, EMMY is a microprogrammable "universal host" with a 200 ns. micro store and an 800 ns. main store (50 and 400 ns. access times, respectively). Both stores are 32 bits wide; 4K words of read/write micro store and 16K words of main store were available during the development of DELtran. Mass storage and intelligent console functions are provided by two cassette tape drives integral to a Data Point 2200 CRT terminal. Unique host characteristics impacting the design of DELtran are:

1) Register, control, and main stores are functionally partitioned; i.e., different micro orders must be used to access each of these stores.

2) All storage resources are 32 bits wide, and may be addressed on a 32 bit word basis -- main store alone may be treated as an 8, 16, 24, or 32 bit wide memory, and addressed on 8, 16, or 32 bit boundaries.

3) EMMY's flexible field extraction operators, which include double shift, are comparatively slow -- consuming as much time as two or three arithmetic or logical operations.

4) Decisions must be implemented by explicit test and branch sequences since EMMY has no implicit tagging capability; more time is needed to determine whether an address refers to main or micro store than is needed to perform a main store access.

5) The basic addressing mode is zero offset register indirection (i.e., the effective address is the contents of a micro register); multi register and/or offset indexing is not available.

A few logistic complications also affected our design. During coding and testing of the DELtran executor, only nascent program support facilities and I/O substructure were available. As a result, only a minimal interface to the external world has been implemented -- all input and output is done through the basic front panel display and control unit, for example.

The "block access unit" anticipated in Neuhauser [9], which was to asynchronously control memory-to-memory transfers, has been supplanted by the "main memory control unit" described in Neuhauser [10]. The earlier design permitted a single command to invoke fully overlappable transfer of an entire multi word block either within or between storage resources; the later design permits only single word transfers between different resources, at a per transfer cost of about 500 ns. in non-overlappable execution time. Because of this, the invocation mechanisms described below differ somewhat from the idealized versions discussed in Hoevel and Flynn [1].

User Influence:

The intended user community is assumed to be composed of general purpose, scientific programmers. User characteristics most relevant to the design of DELtran are:

1) About half the statements in a typical source program deal with program control, and about half are assignment statements (Wichman [11], Rossman [12], and Lunde [13]).

2) The single, most frequent type of statement is "A = B", followed at some distance by "A = A + B" (Knuth [14]).

3) DO statements almost always use an implicit increment (stepping value) of one (Knuth [14], Rossman [12]).

4) Three distinct branches are usually specified for the arithmetic if statement (implied by the distribution of branch statements noted in Flynn [15]).

While these assumptions appear applicable to a variety of user communities and source languages, specific programs could deviate from the implied statistical distribution of operators, names, etc. A more detailed behavioral model could, of course, be extracted from installation-specific trace-tape data.

2. General Description:

Due to the sequential nature of FORTRAN, both at the source and machine code level, a linear outer form is used. The natural scope of definition for source level identifiers is the program or subprogram -- i.e., MAIN, SUBROUTINE, or FUNCTION blocks. Indeed, the lack of any other structured control units leaves little choice in this matter, especially in light of our intent to minimize compilation complexity.

Individual DELtran instruction units are broken down into independently encoded subfields, of varying size, called syllables. Three classes of syllables were required: operand syllables, which denote DELtran variables (or labels); operator syllables, which denote transformation rules to be applied to the DELtran data store; and formlate syllables, which denote initializations to be performed in preparation for a deferred operator syllable ("formlate" is coined from the familiar terms format and template, and combines their respective connotations of semantic and syntactic specification).

Word boundaries may be crossed immediately before or immediately after either operator or formlate syllables: i.e., sequences of operand syllables must lie within a single word. (operand lists for n-ary immediate operators such as CALL, READ, and WRITE excepted). These syllables may be combined in three general syntactic sequences to form DELtran instruction units:

Leading Operator:	<OP> [<A> [<B> [...]]]
Leading Formlate:	<F> [<A> [...]] <OP>
Compound:	<F> [<A> [...]] <OP> [<D> [...]]

Leading operator forms generally deal with program control, involving functions that do not fit well within the familiar molds of binary or unary operators (the leading MOVE operator is an exception; it is coded in this form because of its high frequency of occurrance). The leading formlate construction factors out the operand decode and fetch

computations required by common operator functionalities: diadic (two arguments, one result); monadic (one argument, one result), and onadic (no arguments, no result). The compound form is used only with a few high order functionality operators, or with array access primitives that require information about explicit operand references not provided by the standard formlate interface. The normal sequence of interpretation is for leading formlate constructions:

Decode leading syllable -- extract 5 bit leading syllable from the current instruction word (IW); and transfer control to the appropriate interface routine.

Form interface -- extract all W bit operand reference syllables; fetch values of of arguments; compute address of result, if any.

Decode operator -- extract operator code, and transfer to appropriate semantic routine.

Execute -- compute designated transformation; store result, if any; and begin another cycle of interpretation.

The leading operator form bypasses the explicit operand fetch and interface formation steps, proceeding directly to the execution of the designated function. In this case, the appropriate semantic routine assumes responsibility for fetching, decoding, and accessing (any) operand references. This is similar to the manner in which deferred operators that require additional operands fetch, decode, and access referands identified by deferred operand syllables.

The mechanism for communicating information between interface and semantic routines consists of three micro registers: P, Q, and R. For binary formlates, P will contain the value of the left argument, Q the value of the right argument, and R the address of the result. Lower functionality requirements are derived from this standard

interface by deleting specifications. In the unary case, for example, Q contains the value of the only (and hence still right most) argument, and R the result address.

This "PQR" interface has meaning only within the interpretation of a leading formlate or compound type of instruction unit. Some residual control information, called the DEL program state vector, must be maintained across instruction interpretations, however. The internal DELtran program state is defined by:

1) Instruction Word (IW): a buffer for the DELtran instruction stream.

2) Instruction Pointer (IP): a pointer to the next word of instruction units in the DELtran program store.

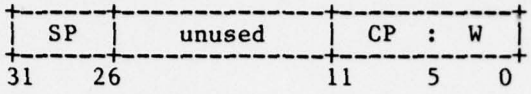
3) Control Pointer (CP): a pointer to a linear definition table for all accessable labels, variables, constants, etc.

4) Stack Pointer (SP): a pointer to the top of a dynamic evaluation stack.

5) Syllable Width (W): a specification for the number of bits in an operand reference syllable.

6) Evaluation Stack (ES): a LIFO queue containing the results of intermediate computations.

Five of these six entities are encoded in three micro registers; the current instruction word is kept in micro register I, and the current instruction pointer is kept in micro register IP. The control pointer CP, the current stack pointer SP, and the current syllable width W are all encoded in a single micro register S:



This assignment leaves four micro registers available for general use. Three of these (P, Q, and R) are temporarily dedicated to the "PQR" interface when interpreting leading formlate or compound instruction forms; but may be reassigned when the standard interface is not required. The remaining micro register, X, is used for general purpose indexing and scratch storage.

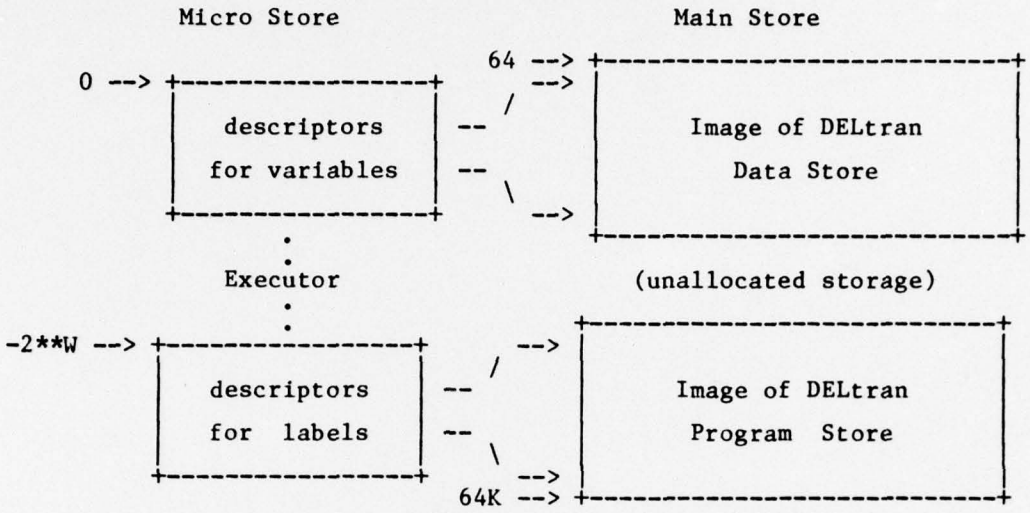
The association between DELtran operand references and referands in the data or program stores is defined by a single linear table called the current contour. Each element of this table, called a descriptor, contains two pieces of information -- a shape and a locator. Shape specifiers (high 8 bits) define an entity's size, justification, and the granularity of its locator, but not its logical data type in the classical sense. Locators (low 24 bits) are directly the address of a referand in EMMY's main store.

The current contour is physically divided into two parts: a data table located at the bottom of micro store; and a label table located at the top of micro store. Since the current contour is always located in a fixed position, a dynamic environment pointer (i.e., the ep in Johnston's Contour Model [19]), is not required -- the control pointer serves as an environment pointer for CALL and RETURN, but is not normally used to interpret DELtran reference codes.

Because it is possible to distinguish between references to variables and references to labels syntactically (for the given FORTRAN source language), judicious placement of descriptors can reduce the number of bits required in operand syllables. An operand reference code N denotes the descriptor at location N if it refers to a variable, and the descriptor at location -2**W+N if it refers to a label -- where W is the number of bits in an operand reference, and micro store is treated as a circularly addressable memory. This means that W may in fact be the least integer such that there are less than 2**W distinct labels and less than 2**W distinct variables, rather

than the least integer such that there are less than 2**W distinct entities (both labels and variables) in a given scope of definition. This addressing scheme is illustrated below:

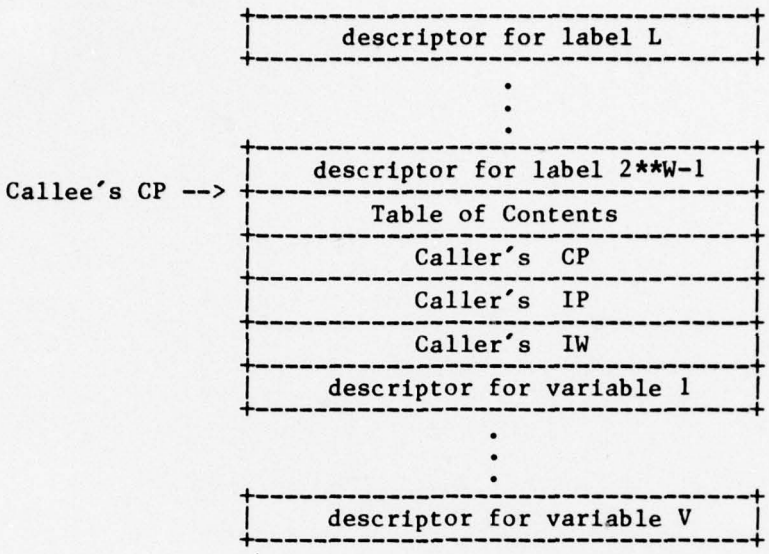
DELtran Reference Structure



This figure also illustrates the general layout of DELtran programs in EMMY's main store; with COMMON and LOCAL storage allocated just above the 64 word evaluation stack, and program modules allocated at the upper end of main store. If more than one procedure is included in a module, COMMON is extended toward the higher addresses and LOCAL for the n+1th procedure is allocated just above that for the n-th procedure (MAIN is the 1st procedure). The actual bodies and skeletal contours for procedures are allocated beginning at the high end of main store and moving toward the lower addresses. This is identical to the storage allocation strategy used by McClure [7], except for an inversion of addresses and the fact that we limit our evaluation stack to 64 elements.

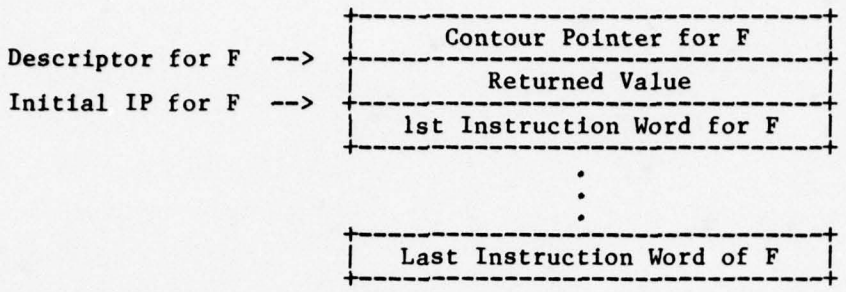
The current contour is initialized by the CALL and RETURN operators from skeletal contours pre-allocated during compilation. There is one skeletal contour for each seperate scope of definition; i.e., for each SUBROUTINE or FUNCTION (including MAIN). Each skeletal

contour consists of a label definition table, linkage area, and a data definition table:



The table-of-contents word defines the number of formal parameters, dynamic (overlay) variables, static variables, and label descriptors for the associated block. The "Caller's ..." words in the linkage contain the DELtran program state vector elements that must be restored upon encountering a RETURN instruction. Skeletal contours are themselves identified by the "-1th" word of a DELtran module; the "0th" word contains the returned value, if it is a FUNCTION; while the "1st" word is the actual beginning of the executable code for the module:

Layout of a DELtran Module (F)



Letting the descriptor for a FUNCTION module identify the referand of its returned value, as well as its entry point, helps to minimize the number of distinct entities in a given scope of definition.

3. Syllable Descriptions:

All DELtran instruction units begin with a 5 bit leading syllable. The 32 distinct codes for this key syllable specify either an immediate operation or a formlate that describes the preliminary processing required to establish the standard interface for a deferred operator.

Five Bit Lead Syllable Encoding

Code	Immediate Syntax	Immediate Semantics
00000	FETCH	fetch next instruction
10000	MOVE <A> <B>	b := a
01000	_TT <OP>	t := OP(t)
11000	_AB <A> <B> <OP>	b := OP(a)
00100	_TA <A> <OP>	a := OP(t)
01100	_AS <A> <OP>	s := OP(a)
10100	_AA <A> <OP>	a := OP(a)
11100	___ <OP>	execute OP
00010	UTU <OP>	u := OP(u,t)
00110	UTA <OP>	a := OP(u,t)
01010	ATT <A> <OP>	t := OP(a,t)
01110	TAT <A> <OP>	t := OP(t,a)
10010	ABS <A> <B> <OP>	s := OP(a,b)
10110	ABC <A> <B> <C> <OP>	c := OP(a,b)
11010	TAB <A> <B> <OP>	b := OP(t,a)
11110	ATB <A> <B> <OP>	b := OP(a,t)
00001	ABA <A> <B> <OP>	a := OP(a,b)
00011	ABB <A> <B> <OP>	b := OP(a,b)
00101	ATA <A> <OP>	a := OP(a,t)
00111	TAA <A> <OP>	a := OP(t,a)
01001	AAS <A> <OP>	s := OP(a,a)
01011	AAB <A> <B> <OP>	b := OP(a,a)
01101	AAA <A> <OP>	a := OP(a,a)
01111	CALL n <F> <A1> ... <An>	invoke F(A1, ..., An)
10001	RETURN	return from invocation
10011	GO <L>	goto l
10101	CGO <I> <L>	goto (l+i-1)
10111	IFE <E> <L>	goto (l+(e=0)+2*(e>0))
11001	IFT <L>	goto (l+(t=0)+2*(t>0))
11011	ENDO <N> <I> <M> <L>	goto l if n=n+i < m
11101	END1 <N> <M> <L>	goto l if n=n+1 < m
11111	BREAK	trap to monitor function

This listing, which is in "trailing zeros" order, uses the same general notation as in the formlate discussion in Hoevel and Flynn [1]. Generic syllables are enclosed in angle brackets "<>", while specific codes are not; a three character mnemonic system is used to

identify formlate structure.

The first letter designates the operand to be associated with the left argument of a binary operator; the second letter designates the operand to be associated with the right argument of a binary (or only argument of a unary) operator; and the third letter designates the operand to be associated with the result.

The following character code is used to signify particular operand bindings. A, B, and C denote the explicit reference codes appearing in the first, second, and third explicit operand syllables following a formlate; the variables identified by these codes are designated by the corresponding lower case letters. S, T, and U denote the top elements of an implicit evaluation stack; T corresponds to the current top of this stack, U to the position just below T; and S to the (unused) position just above T. Again, lower case letters denote the values of these referands. An underscore '_' denotes an argument or result position that is not used.

In practice, lead syllable codes are extracted from the residual instruction word register (I) using the double shift technique, and then added to the microprogram counter ($) to effect an indexed branch. In EMMYXL notation (Hedges [16]):

```
X := 0                          .clear index register X
   .
   .                            .possible intervening code
   .
X,I << 5 ;  $ = $+X             .extract lead syllable
{table of entry points}
```

Instruction units in the table following the extraction may perform useful computations as well as transfer microprogram control to the remaining body of the appropriate routine (due to the semihorizontal nature of EMMY's native language; see Neuhauser [10]).

The DELtran formlate set permits full exploitation of repeated operands (either as arguments alone, or in combination with the result specification), and is "transformationally complete" in the sense that any binding of explicit operands (i.e., primitive variables) and implicit operands (i.e., stack elements) can be generated by local

combinatorial analysis of the FORTRAN source code (Flynn [3], and Hoevel and Flynn [1]). Note also that deferred operators are partitioned disjoint classes according to their functionality by the inclusion of distinct binary, unary, and nullary formlates; and that reverse forms of deferred operators are not needed, since all required argument permutations are contained in the formlate set.

The MOVE operator simply transfers the value of the referand identified by operand reference <A> to the referand identified by operand reference <B>. Simple program control operators such as GO, CGO, IFT, and IFE cause the current instruction word register (I) to be reloaded from the bit address in DELtran's program store identified by the appropriate label descriptor. The single label reference appearing in the CGO, IFT, and IFE constructs is actually the first entry in a subtable of the current contour; the data dependent index into this table is determined by the semantic routine for each of these operators.

ENDO and END1 operators cause the value identified by <N> to be incremented and then execute a GO <L> if the result is less than or equal to <M>. The increment value is assumed to be one for the END1 operator, but is explicitly denoted by <I> for the more general ENDO operator. Breaking out the special case of END1 is indicated not only by the default specification rule for FORTRAN looping constructs, but also by emperical user statistics (Knuth [14]).

CALL and RETURN operators are somewhat more complicated, since they involve modification of the internal state of the DELtran executor. CALL causes the volatile portion of the current contour to be paged out to its static image, which is identified by the control pointer CP. The instruction pointer, instruction word, and status registers (IP, I, and S) are also saved in a linkage area within this skeletal contour, thus saving the DELtran program status vector and hence all information needed to resume the caller's process. The skeletal contour for the callee is then moved into the current contour, and descriptors for formal parameters copied into the appropriate locations. The IP is set to point to the first word of

the callee's program body, the first instruction word is fetched, and the state register S is loaded with the callee's syllable width and control pointer.

RETURN simply undoes a CALL. Only those descriptor elements in the original caller's skeleton contour which were overlayed during the CALL operation need be restored, however. These are easy to determine by comparing the upper and lower reference indice bounds for both programs, which are stored in a linkage area in their skeleton contours. We save and restore the contents of the caller's old instruction word register to avoid wasting static space in the DELtran program store; the time required to perform this linkage is greater than that which would be required simply to fetch a new instruction word from the program store.

Operand Syllables:

As noted above, the width of operand syllables may vary from one scope of definition to another. The current number of bits in an operand syllable, W, is maintained in the low order six bits of the DELtran secondary state register, S, which is automatically saved and restored by the execution semantics for CALL and RETURN. For short subroutines or functions, only three or four bits are needed to identify a unique variable; in larger modules, however, six to eight bits may be needed.

The map from reference codes to descriptors for DELtran variables is simple and direct: the descriptor for variable with reference code N is located at address N in micro store. It is possible to extract an operand reference and look up the corresponding descriptor in a single EMMY instruction, which would appear in the EMMYXL notation as:

$$X,I << S \; ; \quad R = M(X)$$

where X is a previously cleared index register, I is the current instruction word register, and R is a micro register that is to

contain the descriptor value. The low order bits of micro register S, which contain the current value of W, indirectly govern the extent of the double shift from I into X indicated in the first half of the instruction. The second half of the instruction causes the micro store word at the location indicated by the low order twelve bits of the index register to be loaded into R.

The map from reference codes into label descriptors is somewhat more complicated to explain, but equally easy to calculate: the descriptor for the label whose reference code is L is located at -2**W+L, viewing micro store as a circularly addressed memory (the absolute address is 4095-2**W+L, but since EMMY's hardware ignores the upper 20 bits of a micro store address, our circular model is valid). The same micro instruction used to associate variable reference codes with variable descriptors can be used for labels, although the index register X must be initialized to minus one.

Descriptors for variables consist of an 8 bit shape specification in the high order byte, which is actually a command code to the memory control unit that specifies the width of the entity in question (8, 16, 24, or 32 bits), together with a 24 bit locator in the low order bytes. The shape designator also specifies the granularity for the locator (8, 16, or 32 bit quanta); the locator directly identifies the main store image of a referand in the DELtran data store.

Descriptors for labels are similarly structured, but in this case the locator is actually a bit address in the DELtran program store. The target instruction word must be shifted after loading to obtain proper alignment. The granularity specifiers within the shape code are used to minimize the magnitude of this shift.

Deferred Operator Syllables:

Deferred operators are categorized as diadic (two arguments, one result), modadic (one argument, one result), or onadic (no arguments, no results). Data types are not checked dynamically because FORTRAN

is such a strongly typed language in its own right, and hence distinct operator codes are used to denote integer and floating functions. Some collapsing of the DEL operator set was possible where only the sign of an operand or equivalence to zero need be checked, as with the IF statement, since these representations are the same for both fixed and floating point (internal value representation consistent with the 370 architecture has been used for pragmatic reasons; see Wallach [17]).

Deferred operator syllables are decoded in the same manner as leading syllables, except that different branch tables are used (one for 4 bit binary operator codes, one for 4 bit unary operator codes, and one for 3 bit nullary operator codes).

Four Bit Encoding of Diadic Operators

Code	Deferred Syntax	Deferred Semantics
0000	FETCH	fetch the next instruction word
1000	A2E <D>	associate D with (p,q)-th element of r
0100	F+	r := p+q (floating add)
1100	I+	r := p+q (integer add)
0010	F-	r := p-q (floating subtract)
0110	I-	r := p-q (integer subtract)
1010	F*	r := p*q (floating multiply)
1110	I*	r := p*q (integer multiply)
0001	-A2-	prefix for array accessing operators
" 00	MA2 <D>	r(p,q) := d
" 01	A2M <D>	d := r(p,q)
" 10	TA2	r(p,q) := t
" 11	A2S	s := r(p,q)
0011	F/	r := p/q (floating divide)
0101	I/	r := p/q (integer divide)
0111	F^F	r := p**q (floating to floating power)
1001	I^I	r := p**q (integer to integer power)
1011	FST	r := sgn(p)*q (floating sign transfer)
1101	IST	r := sgn(p)*q (integer sign transfer)
1111	BREAK	trap to monitor

The -A2- operators are perhaps not self defining; in general, the two argument values in the P and Q interface registers to be treated as the first and second subscripts for the array whose descriptor will be in the result register, R. A2E causes the effective address of the indicated array element to be computed, creates a descriptor to this element by combining the shape field from the array descriptor with this address, and stores the result in the contour slot for the deferred reference code D. MA2 and A2M operators work in a similar

fashion, but actually cause a state transformation in the DELtran data space -- they are similar to the MOVE operator. TA2 and A2S are "push" and "pop" operators that transfer values between the evaluation stack and array elements.

Initially, we intended to perform array accessing implicitly by dynamically checking the structural type of each variable descriptor before using it to load or store a value. However, without specific hardware support this proved to be too inefficient. Substantial code compaction, as well as execution time reduction for array accesses, may be possible in systems based on a tagged architecture host, such as described in Feustel [18] and Iliffe [4].

Bounds checking is not performed, following with the tradition established by IBM. It would be easy to incorporate by modifying the appropriate array accessing routines, and would not involve a high space or time penalty for the EMMY host. The multiplier needed to compute the effective address of an indexed array element is stored at the "base" of the array (i.e., is its zero-th element; this works for FORTRAN since array subscripts must begin with one).

Four Bit Encoding of Modadic Operators

Code	Deferred Syntax	Deferred Semantics
0000	FETCH	fetch new instruction word
1000	AlE <D>	associate reference code D with r(p)
0100	FLOAT	r := float(p)
1100	FIX	r := fix(p)
0010	F~	r := -p (floating negate)
0110	I~	r := -p (integer negate)
1010	LOG	r := log(p) (logarithm)
1110	SIN	r := sin(p) (sine)
0001	-Al-	prefix for array accessing operators
" 00	MAl <D>	r(p) := d
" 01	AlM <D>	d := r(p)
" 10	TAl	r(p) := t
" 11	AlS	s := r(p)
0011	COS	r := cos(p) (cosine)
0101	TANH	r := tanh(p) (hyperbolic tangent)
0111	PAUSE	pause with code p
1001	STOP	stop with code p
1011	TIME	r := (current time)-p
1101	not used	
1111	BREAK	trap to monitor

The "A1" array oriented operators are just like the "A2" operators described above, except that only a single subscript is required (the argument value in the P register of the standard interface). Again in the IBM tradition, no bounds checking is performed. The TIME function, although not required by the ANS specification [8], is included in order to facilitate experimental evaluation of the final system.

Some compression of the operator set suggested by the semantics of Basic FORTRAN has been obtained by noting a few non-trivial algebraic relations. In particular, the EXP function can be replaced by the binary F^F operator -- i.e., instead of generating:

_XY [<A> [<B>]] EXP

we generate:

AX'Y' <e> [<A> [<B>]] F^F

(where X'Y' is derived from XY by transforming A->B and B->C). This is the same as the observation that EXP(x) can be rewritten as E**(x), where E is a constant with value 2.718..., for any expression x.

Three Bit Encoding of Onadic Operators

Code	Deferred Syntax	Deferred Semantics
000	FETCH	fetch next instruction word
100	SET <U> <F>	set Unit = U and Format = F
010	READ n <D1>...<Dn>	input to D1...Dn as per Unit/Format
110	WRITE n <D1>...<Dn>	output from D1...Dn as per Unit/Format
001	REWIND	rewind Unit
011	BACKSPACE	backspace Unit
101	ENDFILE	write end-of-file mark on Unit
111	BREAK	trap to monitor

Compound instruction units of the form "___ <onadic OP> ... " are really nothing more than a partial frequency encoding of infrequent and/or difficult to handle functions, the bulk of which deal with input output. Two residual control cells, Unit and Format, are used to maintain the status of I/O operations. Unit corresponds to a logical designation of a specific file/device/channel combination, and would in practice be bound by a surrounding operating system as specified by some external job control language. The Format cell is merely a byte pointer into a string of field specifications produced

during compilation from the appropriate FORTRAN format statement.

Encoding of Format Control (I/O) Operators

FORTRAN Construct	DELtran Construct Symbolic	Actual
n(	(n	0 n
Fw.d	F w d	1 w d
Ew.d	E w d	2 w d
In	I n	3 n
nX	X n	4 n
//.../ (n slashes)	/ n	5 n
nHab... (n chars.)	H n a b ...	6 n a b ...
n... (repeat count)	REP n	7 n
)	)	8

Although the full I/O structure indicated above has not yet been implemented, the intent is that it should proceed as a subinterpretation, either with EMMY performing conversions under control of the current Format, or with the control device for Unit performing these conversions asynchronously. The Unit and Format residual control cells are, respectively, the environment pointer and instruction pointer for this subinterpretation. An entire byte is used to encode formatted field specifications simply to keep this process as simple as possible; the spatial penalty is low since I/O statements are statically insignificant.

4. Examples:

A few examples may help clarify the preceeding discussion:

	FORTRAN Statement		DELtran Equivalent
1)	`A = B`		`MOVE <B> <A>`
2)	`I = J-I`		`ABB <J> <I> I-`
3)	`I = J*J + I`		`AAS <J> I*` `TAA <I> I+`
4)	`GOTO 10`		`GO <#10>`

5)	`DO 10 I = 1, 100`
	`A = F(A,I)`
	`10 CONTINUE`
		`#10`	`MOVE <1> <I>`
			`CALL 2 <F> <A> <I>`
			`MOVE <F> <A>`
			`END1 <I> <100> <#10>`

6)	`IF (A-B) 1,2,3`		`ABS <A> <B> F-` `IFT <#1>`
7)	`WRITE (6,10) N,M`		`___ SET 6 10` `___ WRITE 2 <N> <M>`
8	`10 FORMAT (1H ,2I5)`	`#10`	`<(> <H> 1 <R> 2 <I> 5 <)>`
9)	`I = A(I,I)`		`AAB <I> <A> A2S` `_TA <I> FIX`

5. References:

[1] Hoevel, Lee W., and Flynn, Michael J., "The Structure of Directly Executed Languages: A New Theory of Interpretive System Support," Technical Note No. 130, Digital Systems Laboratory, Stanford University, Stanford, California, March 1977.

[2] Hoevel, Lee W., "Languages For Direct Execution," Proceedings of the 7th Annual Workshop on Microprogramming (SIGMICRO 7), September 1974, pp. 307-16.

[3] Flynn, Michael J., "The Interpretive Interface: Resources and Program Representation in Computer Organization," Proceedings of the Symposium on High Speed Computers and Algorithm Organization, University of Illinois, Champaign Illinois, (Pub. Academic Press) April 1977.

[4] Iliffe, J. K., "Interpretive Machines," lecture survey notes, Digital Systems Laboratory, Stanford University, Stanford, California, May 1977.

[5] Welin, Andrew M., "The Internal Machine," ACM-IEEE Symposium on High-Level-Language Computer Architecture, University of Maryland, College Park, Maryland, November 7-8, 1973, pp. 91-100.

[6] Heising, W. P., "History and Summary of FORTRAN Standardization Development for the ASA," Communications of the ACM, Vol. 7, No. 10, October 1964, p. 590.

[7] McClure, Robert M., "CUC Basic FORTRAN Description," (private working notes), 1970.

[8] American Standards Association Sectional Committee X3, Computers and Information (R. V. Smith, ed.), "FORTRAN vs. Basic FORTRAN -- A Programming Language for Information Processing on Automatic Data Processing Systems," Communications of the ACM, Vol. 7, No. 10, October 1964, pp. 591-625.

[9] Neuhauser, Charles J., "System Description of the JHU Emulation Laboratory and Host Machine," Proceedings of the 7th Annual Workshop on Microprogramming (SIGMICRO 7), September 1974, pp. 28-33.

[10] - ,"An Emulation Oriented, Dynamic Microprogrammable Processor (Version 3)," Technical Note No. 65, Digital Systems Laboratory, Stanford University, Stanford, California, October 1975.

[11] Wichman, B. A., "Five Algol Compilers," Computer Journal, Vol. 15, No. 1, January 1972.

[12] Rossman, George, "Statistical Usage of the 360 Architecture," Technical Report, Palyn Associates, San Jose, California, 1973.

[13] Lunde, A., "Empirical Evaluation of Some Features of Instruction Set Processor Architectures," Communications of the ACM, Vol. 20, No. 3, March 1977, pp. 143-52.

[14] Knuth, D. E., "An Empirical Study of FORTRAN Programs," Software Practice and Experience, Vol. 1, 1971, pp. 105-33.

[15] Flynn, Michael J., "Trends and Problems in Computer Organizations," IFIPS Congress, Stockholm, Sweden, August 1974 (Pub. North Holland 1975), pp. 2-10.

[16] Hedges, Tomas S., "EMMY/360 Cross Assembler," Technical Note No. 74, Digital Systems Laboratory, Stanford University, Stanford, California, December 1975.

[17] Wallach, Walter A., "EMMY/360 Functional Characteristics," Technical Report No. 114, Digital Systems Laboratory, Stanford University, Stanford, California, June 1976.

[18] Feustel, Edward A., "On the Advantages of Tagged Architecture," IEEE Transactions on Computers, Vol. C-22, No. 7, July 1973, pp. 644-56.

[19] Johnston, John B., "The Contour Model of Block Structured Processes," Proceedings of the SDSPL (SIGPLAN Notices, Vol. 6), February 1971, pp. 55-82.